METHOD AND APPARATUS FOR MANAGING DATA SERVICES IN A DISTRIBUTED COMPUTER SYSTEM

FIELD OF THE INVENTION

[01] This invention relates to management of networked computer systems and to data services such as replication, mirroring, and data "snapshots" and, in particular, to distributed management of such data services.

BACKGROUND OF THE INVENTION

[02] It is common in many contemporary computer systems to provide administrative personnel with data services that allow data storage to be controlled and backup copies of stored data to be made where necessary. These data services can include data volume management in which one or more logical "volumes" are created from stored data so that the data in each volume can be manipulated as a unit.

[03] Data backup services may include data replication in which a second copy or "mirror" of information located at a primary site is maintained at a secondary site. When changes are made to the primary data, updates are also made to the secondary data so that the primary data and the secondary data remain "synchronized."

[04] Another data backup service that can be provided is data imaging in which a data copy often called a data "snapshot" or "image" can be made of stored data at a designated point in time. This service provides a system administrator with the ability to make and to maintain the aforementioned replicated data storage systems.

[05] Still other data services can be provided. For example, data stored and retrieved in a data storage system may be transferred through a fast cache memory in a process called "data caching" in order to improve overall data storage performance. Often it is desirable for system administrators to be able to control whether data caching is used with a particular data storage or a data volume residing on that storage in order to effectively use a limited resource.

[06] No matter which of the data services are used, a significant amount of management time can be consumed in initially setting up the data service and

managing it after it is running. For example, management of each of the aforementioned data services requires the ability for a manager to discover volumes existing in the system, verify that those volumes are suitable for use with a particular data service and configure the volumes if they are not suitable.

[07] In a large, distributed computer system connected by a network, management personnel and resources may be physically located anywhere in the system. However, the data manipulation processes, which actually perform the data services, are typically low-level routines that are part of an operating system kernel running on a particular machine. These routines typically must run on that machine and are written in a platform-dependent language. Thus, prior art systems required a manager to physically log onto each local host in a distributed system in order to manage data services. Further, there was no way to diagnose problems that occurred when several data services were using the same volume.

SUMMARY OF THE INVENTION

[08] In accordance with the principles of the present invention, management of data services is provided by a three-tiered system. The lowest tier comprises management facade software running on each machine that converts the platform-dependent interface written with the low-level kernel routines to platform-independent method calls. The middle tier is a set of federated Java beans that communicate with the management facades and with the upper tier of the system. The upper tier of the inventive system comprises presentation programs that can be directly manipulated by management personnel to view and control the system.

[09] In one embodiment, each data service is managed by a separate federated bean running in the host where the data service is located. The beans export interfaces containing methods that can be called by the presentation programs so that management personnel located at that host can access and control the data services in that host. In addition, an identifier for each bean in the distributed computer system can be retrieved from a distributed lookup service so that, using the identifier, management

personnel located at one host can manage data services located at another host by calling methods in the remote bean.

[10] In another embodiment, a data service station federated bean monitors the distributed lookup service and retrieves and caches the other bean identifiers. Then a presentation program or other user can lookup the data service station bean and quickly discover data services on other hosts and retrieve the appropriate identifier for the beans on that host in order to manage the data services on that host.

[11] In still another embodiment, a data service volume federated bean in each host discovers data volumes on that host and controls the volumes via a data services management facade running on the host. This bean allows a manager to view and configure volumes on that host. The data service volume federated bean can configure and control volumes with either a SCSI terminal emulation interface or a storage volume interface

[12] In another embodiment, a configuration manager bean runs on each host and persistently stores configuration information regarding the volumes on that host. The configuration manager federated bean for a host computer is contacted by federated beans that manage each of the data services when a data service starts and stops using a data storage volume associated with the host computer. The data services can access the configuration information during the host system boot process and before file systems are mounted and management software can obtain the same information at a later time in a consistent manner.

[13] In yet another embodiment, a data replication management facade runs on each host and at least one data replication bean also runs on the host. The data replication beans allow a manager located at one host to configure a primary data volume at that host and to configure a secondary data volume at that host or another host in the distributed computer system. A data replication system can then be set up using the bean in the primary host and the bean in the secondary host. Therefore, a user needs to log onto only one machine in order to configure the entire data replication system.

[14] In still another embodiment, a data caching management facade runs on selected hosts and at least one data caching bean also runs on those hosts. The data-caching bean allows a manager to configure an entire data caching system from a single location so that the manager can cache individual volumes "on the fly" during ongoing data processing operations.

[15] In yet another embodiment, a data imaging management facade runs on selected hosts and at least one data imaging bean also runs on those hosts. The data-imaging bean communicates directly with a management GUI or CLI and is controlled by user commands generated by the GUI or CLI. Therefore, a manager can configure the entire data imaging system from a single location.

[16] In another embodiment, the distributed computer system includes an event service to which the federated beans post messages regarding their status and configuration. Another federated bean monitors the event service and sends human-readable messages to administrators in order to inform them that an event in which they may be interested has occurred.

BRIEF DESCRIPTION OF THE DRAWINGS

[17] The above and further advantages of the invention may be better understood by referring to the following description in conjunction with the accompanying drawings in which:

[18] Figure 1A is a block schematic diagram of illustrating the platform-specific kernel drivers that provide a variety of data services in an application server.

[19] Figure 1B is a block schematic diagram of illustrating the platform-specific kernel drivers that provide a variety of data services in a storage server.

[20] Figure 2 is a block schematic diagram of a three-tiered system for providing data services in a single host, illustrating an upper presentation tier, a federated bean middle tier and a management facade lower tier.

[21] Figure 3 is a block schematic diagram illustrating a collection of federated beans in an exemplary system and their interaction.

[22] Figure 4 is a block schematic diagram illustrating the use of a data service station bean to lookup other federated beans.

[23] Figure 5 is a block schematic diagram illustrating the operation of a remote notification bean in notifying administrators of management events in a data service management system.

[24] Figure 6 is a schematic block diagram illustrating how multiple hosts communicate via the federated beans in the middle tier.

[25] Figure 7 is a schematic block diagram illustrating the implementation of a simple data replication system using the principles of the present invention.

[26] Figure 8 is a flowchart showing the steps of an illustrative process for installing data replication software in the system of Figure 7.

[27] Figures 9A-9D, when placed together, form a flowchart showing the steps of an illustrative process for configuring the data replication system of Figure 7.

DETAILED DESCRIPTION

[28] Data Services are software products that consist of two parts: a set of kernel drivers, which provides the actual service on the local platforms, and the user level management software. The kernel drivers would generally by implemented in platform-specific code, for example, in C routines that expose application programmer interfaces (APIs) that can be accessed only from the host in which the layer is installed. The set of kernel drivers providing the service can be installed on application servers as well as dedicated storage servers. These installations are illustrated in Figures 1A and 1B.

[29] As shown in Figure 1A, on an application server 100, the data service kernel modules 108 layer within the operating system I/O stack above volume manager 118 and below the disk device drivers 106. The data service kernel modules include a storage volume module 110 that implements a storage volume (SV) interface data service that provides data redirection. In particular, the storage volume layer 110 insinuates itself between the standard Small Computer Standard Interface (SCSI) block

device driver 106 and the underlying drivers and shunts I/O information through the other data service kernel modules 112-116.

[30] The storage network data replicator (SNDR) kernel module 112 provides data replication services that involve transparent replication of volumes over public or private Internet protocol infrastructure, or locally, via SCSI protocol, over fibre channel connections.

[31] The data imaging module 114 implements a "point-in-time" volume copy data service between volumes in a data image volume set. The data image volume set contains the original logical volume (the master volume), the point-in-time copy of the original (the shadow volume) and a volume used to store a bitmap that tracks the differences between the master and shadow volumes. Once the data image volume set is established, the master and shadow volumes can be accessed independently. The data-imaging module allows data updates to be sent from the master volume to the shadow volume as well as updates to be sent from the shadow volume to the master volume when desired.

[32] The caching module 116 provides block based caching operations for disk input/output. These operations provide typical caching functionality, such as read caching, read ahead and small write coalescing for sequential writes. Module 116 also provides write caching when non-volatile RAM cards are installed as a safe store (called a "fast write cache").

[33] On a dedicated storage server 119 as illustrated in Figure 1B, the kernel modules 122 are located between fibre channel drivers 120 and the volume manager software 132. Modules 122 are accessed through an emulation layer 124 that allows the storage server to appear as a SCSI target to fibre-channel-connected open system hosts. Thus, the SCSI Target Emulation (STE) module 124 provides an STE data service that allows any backend storage to be exported for use on another host through a fiber channel. The host that has the STE kernel module 124 runs a fibre port in SCSI target mode, while the fibre ports at the client run as SCSI initiators.

[34] The network data replicator module 126, the data imaging module 128 and the data caching module 130 operate in the same manner as they do in the

application server example shown in Figure 1A. The data service kernel module architecture requires that any volume that will be used by a data service must already be under the control of either the SCSI Target Emulation (STE) data service module 124, or the Storage Volume Interface (SVI) data service module 110. The difference is that the STE volumes are always exported to remote hosts whereas local volumes must be SVI volumes.

[35] A data services management system constructed in accordance with the principles of the invention comprises three layers or tiers. The first, or upper, tier is a presentation layer with which a manager interacts at a single host location. The upper tier, in turn, interacts with the middle tier comprised of a plurality of federated beans, each of which performs specific tasks in the data services system. The federated beans can communicate with each other both in the same host and in other hosts via a network connecting the hosts. Some of the beans can communicate with the lowest tier that comprises management facades. Management facades communicate to the aforementioned kernel modules that actually perform the data services. In this manner, an entire data service system can be configured and managed from a single location.

[36] Figure 2 shows a host system 200 that illustrates the contents of the three tiers running in a single host. The inventive data service management system comprises three layers or tiers: an upper tier 204, a middle tier 206 and a lower tier 208. The upper tier 204 is a presentation level that can be implemented with either a graphical user interface (GUI) 220 or a command line interface (CLI) 222. A manager interacts with this level, via the GUI 220 or CLI 222, in order to create, configure and manage a data services system.

[37] The middle tier 206 is implemented with a plurality of Federated Java™ (trademark of Sun Microsystems, Inc.) beans. These beans comply with the Federated Management Architecture (FMA) Specification 1.0, a Java technology-based component architecture and management services for automated, dynamic network management developed by Sun Microsystems, Inc. The FMA specification provides a standard for communication between applications, services and devices across a

heterogeneous network, which enables developers to create solutions for complex distributed environments. The FMA Reference Implementation (RI) source code is available at http://java.sun.com/aboutJava/communityprocess/final.html.

[38] The federated beans use a distributed management framework that implements the FMA specification for distributed management of data services. This framework is called the Jiro™ framework (trademark of Sun Microsystems, Inc.) and is developed by Sun Microsystems, Inc. This framework uses the concept of a management domain to provide services. A management domain is a portion of a network with attached managed resources and available management services used to manage those resources. Within a management domain, the framework provides for base and dynamic services. The base services include, a controller service, an event service, a logging service, a scheduling service and a transaction service. The federated Java provides dynamic services beans of the middle tier. Dynamic services require a hosting entity called a "station", which is a mechanism to allow many services to run within a single Java Virtual Machine. Every management domain contains one or more general-purpose shared stations.

[39] In addition, the Jiro™ technology provides a lookup service that is used to register and locate all Jiro™ technology services, including both base and dynamic services, that are available in a management domain. Details of the Jiro™ framework and its use are available in the "Jiro™ Technology SDK Programmer's Reference Manual" available at http://www.jiro.com, which manual is incorporated by reference in its entirety.

[40] For volume management purposes, the main federated bean is the data services volume (DSV) bean 230. DSV Bean 230 is responsible for discovering volumes available on the local system 200, controlling those volumes when necessary via an SVI/STE management facade 240, and coordinating the use of those volumes between other data service federated beans. As discussed below, in the process of managing volumes, the DSV bean 230 uses other beans (not shown in Figure 2) that may be located on the same, or different, machines. In addition, the GUI 220 and the

CLI 222, which are part of the presentation layer 204, communicate with the DSV bean 230 running in the host 200 where the GUI 220 and CLI 222 are running as indicated in Figure 2.

[41] DSV Bean 230 is a Federated Bean as described in the aforementioned Federated Management Architecture (FMA) specification. When created, it registers itself with a local Jiro™ station, and provides its services to any other federated beans within the same Jiro™ management domain. Effectively, DSV Bean 230 provides other federated beans in the management domain acting as clients the ability to explicitly or implicitly control the SVI and STE data services on its host, as necessary.

[42] Along with providing the ability to control the SVI and STE data services, DSV Bean 230 also gives clients the ability to discover what other applications are currently using a particular volume.

[43] The volume managing capability of the invention is actually implemented in the kernel layer 210 shown running in host 200 in Figure 2. In particular, access by the host 200 to a resource 260, which can be a data storage component, is provided by a layered stack 250 comprising the aforementioned SVI or STE layer 252, as appropriate. Application programs running in host 200, such as application 228, and the host file system access resource 260 though the layered stack 250 as indicated schematically by arrow 238.

[44] In order to provide for data service management capability in accordance with the principles of the invention, the SVI/STE layer 252 and the data services layer 254 are controlled by software running on the lower tier 208 of the inventive data services system. The lower tier includes a native interface 242 that converts the APIs exported by the SVI/STE layer 252 into a platform-independent language, such as Java™. A DSV management facade 240 that provides the required remote management capability, in turn, controls the native interface 242.

[45] The DSV management facade 240 provides a means by which the SVI/STE layer 252 can be accessed and managed as a Jiro™ service, i.e., a service that can be managed in a distributed environment from a remote host. The DSV

management facade 244 is essentially an object-oriented model of the kernel-resident SVI/STE layer 252. It provides a collection of APIs to manage the SVI/STE layer 252. The DSV federated bean 230 uses the DSV management facade 240 to configure, control and examine the status of the SVI/STE layer 252 and to provide other important functions.

[46] In this arrangement, the GUIs 220 and 224 or the CLIs 222 and 226 provides the presentation function, the federated beans 230 and 234 implement the management model and the management facades 240 and 244 act as the managed resource. All of the data services will have GUIs or CLIs that operate in a similar, unified environment. Each GUI is responsible for displaying and setting data service information. The attributes that can be set or displayed are retrieved via a proxy to a corresponding federated bean. In order to use a bean, a GUI performs a lookup for the federated bean service, and, if the service is found, the GUI downloads a proxy to that service and uses it to make calls on methods in the bean interface. The bean exports methods from a management facade that, in turn, exports methods from the native interface which mimics the data service. The information retrieved using the proxy is displayed in the GUI or the information the user enters to the GUI is submitted down the stack via the proxy. If the service is not found, an error will be relayed to the user.

[47] Each CLI works in a manner very similar to the GUI with the exception of one step. The CLI is written as a script, which invokes a JAVA application that follows the same protocol used by the GUI to obtain a bean proxy, and then uses that proxy to call bean methods. In addition, each time the user enters a command, a new proxy is created, used, and destroyed.

[48] Other data services are implemented and controlled in a similar manner. Illustratively, data services (DS) bean 234 implements one of the aforementioned data services and also communicates with the DSV bean 232, whenever bean 234 starts or stops using a volume managed by DSV bean 230. Management personnel can control DS bean 234 by means of a GUI 224 or a CLI 226. In order to manage a data services system, DS bean 234 communicates with a data services layer 254 in the layered stack 250, via a data services management facade 244 and a native interface 246. As with

the corresponding DSV management facade 240 and native interface 242, the data services management facade 244 provides a means by which the data services layer 254 can be accessed and managed as a Jiro™ service. The native interface 246 converts the platform-specific kernel routine API's to platform independent interfaces. In a similar manner other data services can be implemented with other layers represented by layer 256. The native interfaces and management facades corresponding to native interfaces 242 and 246 and management facades 240 and 244 are not shown in Figure 2 to simplify the drawing.

[49] Whenever changes are made in the data configuration of host 200, both the DSV bean 230 and the DS bean 234 can inform a configuration manager bean (not shown in Figure 2) of the change in configuration information. DS bean 234 can also retrieve configuration information from the configuration manager bean under appropriate situations. The configuration manager bean maintains a persistent view of the configuration of the data services system on host 200. In this manner, if the host is interrupted during an operation, it can be restored to the proper state when the operation is resumed.

[50] Figure 3 illustrates a collection of federated beans as might be running in an exemplary host computer. This collection represents the middle tier of such a system. The corresponding upper and lower tiers have been omitted for clarity. A client, such as a GUI or CLI, can use these beans by obtaining a handle, or identifier, to a bean from a lookup service, such as the Jiro™ lookup service. Once the handle is obtained, it can be used to access an interface exported by the bean and call appropriate methods in the interface. The middle tier 300 comprises a data-imaging bean 302 that manages a "point-in-time" volume copy data service between volumes in a data image volume set. The bean 302 allows a manager to configure, via a GUI or CLI (not shown in Figure 3), the master, shadow and bitmap volumes and to set up a volume copy between the master and shadow volumes. Other administrative tasks can also be accomplished via the bean 302.

[51] A storage cache manager bean 304 manages caching operations for selected disk volumes, including read caching, read ahead and small write coalescing for sequential writes. The bean 304 allows system administrators to engage or disengage the caching operations "on the fly" to allow optimum system usage. Bean 304 also gathers cache statistics to allow system administrators to monitor caching operations and controls non-volatile RAM cards to enable write caching.

[52] Storage network data replication bean 310 manages a data replication system that replicates a primary volume to a secondary volume over a network. The bean 310 allows system administrators to select and configure the primary and secondary volumes and to set up and start the data replication system and monitor its progress.

[53] Whenever beans 302 and 310 start and stop using a particular volume to provide the data services they control, they contact the data service volume bean 306 and use it to manage the volumes that they use as indicated schematically by arrows 316 and 322. In this manner, the data service volume bean 306 can keep track of volume usage and, in some cases, can provide information to a requesting bean regarding the identity of other services using a particular volume and the type of usage.

[54] Whenever changes are made in the configuration of the data services and data volumes controlled by beans 302, 304 306 or 310, the beans can inform a configuration manager bean 308 of the change in configuration information or request configuration information as indicated schematically by arrows 318, 320, 324 and 326. Configuration manager bean 308 maintains the overall configuration of the data services on the host on which it resides and persists this information by means of a direct interface with a storage system. In this manner the information is available at boot up so that the system can be restarted after a power down or failure and is also available to management personnel during the normal operation of the system.

[55] Two additional beans, a remote notification bean and a data service station bean are also provided. As will be discussed below, the remote notification bean monitors events generated by the other beans and posted to an event service, such as the Jiro™ event service and reports occurrences of event in which an administrator has

indicated interest by a human readable notice, such as an e-mail. The data service station bean 314 assists the data services beans in obtaining handles or identifiers to other beans by subscribing to the Jiro™ lookup service to obtain the bean handles and then caching the handles. Consequently, when a bean needs to obtain a handle to another data service bean, it can obtain the handle from the data service station bean.

[56] Figure 4 illustrates, schematically, the operation of the data service station bean. In this figure, a client, such as GUI 402 uses the data service station (DSS) bean 404 to obtain a handle to another Data service (DS) bean 406. To obtain the DS bean handle, the GUI 402 sends a lookup request for the DSS bean 404 to the Jiro™ lookup service 400 as indicated by arrow 408. The lookup service 400 returns the DSS bean handle to the GUI 402 as indicated by arrow 410.

[57] Next, using the DSS bean handle, the GUI 402 makes a call on a method in the interface 405 of the DSS bean 404 to request the handle of the DS bean 406. This call is illustrated by arrow 412. The DSS bean 404 returns the DS bean handle from its internal cache memory as indicated by arrow 414. Finally, using the DS bean handle retrieved from the DSS bean, the GUI can call methods in the DS bean interface 407 as illustrated by arrow 416.

[58] The operation of the remote notification bean 312 is illustrated in Figure 5. During operation, the data service beans, including the data imaging bean 502, the data service volume bean 504, the storage cache manager bean 506 and the storage network data replication bean 508 generate event messages and forward the messages to the Jiro™ event service 500. For example, these messages may be generated when a data service configuration changes, when an error is encountered or when a process finishes. When it receives these event messages, the Jiro™ event service sends notifications to the remote notification bean 512 which is registered with the event service 500 as a listener 510. The remote notification bean 512 then generates a human readable message and forwards the message to a system administrator via a conventional delivery system, such as an e-mail system 514.

[59] In addition to controlling data services on their own host as described above, the federated beans can communicate via a network connecting the hosts to

control data services on other hosts. For example, Figure 6 illustrates a system with three hosts 600, 602 and 604 connected by a network 603. Although each of hosts 600, 602 and 604 in this figure include the components illustrated in Figure 2, only the middle tier has been shown in detail in each host 600, 602 and 604 in order to simplify the diagram.

[60] Host 600 has an upper tier 606, a middle tier 608 and a lower tier 610. A GUI/CLI 624 running in the upper tier 606 of host 600 can be used by a manager to create and manage a data service, such as a storage network data replication (SNDR) system. A manager could also manage the system from a similar GUI/CLI running in upper tiers 612 and 618 of hosts 602 and 604, respectively.

[61] GUI/CLI 624 interacts with an SNDR federated bean 626 running in host 600. As discussed with respect to Figure 2, bean 626 can interact with a configuration manager bean 628 and a data service station bean 630 also running in the middle tier 608 of host 600. In a similar manner, an SNDR federated bean 634 is running in the middle tier 614 of host 602. Bean 634 can interact with a configuration manager bean 632 and a data service station bean 636 also running in the middle tier 614 of host 602. Another SNDR federated bean 640 is running in the middle tier 620 of host 604. Bean 640 can interact with a configuration manager bean 638 and a data service station bean 642 also running in the middle tier 620 of host 604.

[62] Although a manager interacts with the GUI/CLI running in host 600, an exemplary data replication system can be created involving volumes located in two or more hosts. In order to provide communication between hosts, the SNDR beans in the various hosts can communicate with SNDR beans and data services beans in other hosts, via the network 603. For example, bean 626 can communicate with the data service station bean 630 on its host to ascertain the location of data service beans running on other hosts. Once the location of these data service beans is known, bean 626 can interact with the data service station beans running on these other hosts in order to retrieve federated bean proxies of other SNDR beans that are running on the other hosts. The federated bean proxies are identifiers or "handles" that can be used by bean 626 to call appropriate methods in other SNDR beans, whose methods are

necessary to perform different operations. For example, bean 626 can contact data service station bean 636 on host 602, via network 603, as indicated schematically by arrow 652, in order to receive a federated bean proxy to SNDR bean 634. Using this proxy bean 626 can then communicate directly with bean 634 by calling methods in bean 634 as indicated schematically by arrow 650.

[63] In a similar manner, bean 626 can contact data service station bean 642 on host 604, via network 603, as indicated schematically by arrow 654 in order to receive a federated bean proxy to SNDR bean 640. Using this proxy, bean 626 can then communicate directly with bean 640 by calling methods in bean 640 as indicated schematically by arrow 656.

[64] Using federated bean proxies, as appropriate, a manager who controls bean 626 can also control SNDR beans running in other hosts to configure, control and monitor the SNDR data service layers in those hosts, via a management facade and a native interface as previously discussed, in order to locate, prepare and manage volumes in those hosts. This arrangement gives an administrator working with GUI/CLI 624 the capability of remotely managing data replication as opposed to the prior art mechanism of purely primary and secondary host-based management. When an SNDR federated bean encounters an error in a remote operation carried out on a proxy it retrieved earlier, it retrieves the proxy again and retries the operation before throwing an error. This retry capability allows a restart of other services that are necessary for the data replication system to perform its functions.

[65] An example data services system that performs data replication and is managed in accordance with the principles of the invention is illustrated in Figure 7. Figure 8 illustrates the steps performed in initially configuring the system. Figures 9A-9D show a flowchart illustrating the steps carried out by a data replication system using the principles of the invention to set up the configuration shown in Figure 7 and to remotely replicate data from volume A (720) of host system A (702) located in geographic region I to volume B (732) of host system B (722) located in a different geographic region II using the inventive data service management software, via network 710.

[66] In order to use the inventive system, the software that is required must first be installed in the system. The steps of the installation process on one host (for example, host system A in geographic region I) are shown in Figure 8. The installation process begins in step 800 and proceeds to step 802 where the data services software, such as that used by the SNDR system, is installed on host system A 702. This software includes the data service volume layer software 252 and a data services layer, such as a network data replicator layer 254. Other data service layers, such as a cache layer, can also be included in this installation process.

[67] Next in step 804, the Jiro™ software is installed. The installation process for this software is explained in detail in the aforementioned Jiro SDK.

[68] In step 806, the data service management software is installed. This software includes an SNDR management facade 244 and the native interface 246. It also includes a SNDR federated bean 234 and the command line interface 224 or graphic user interface 226, as appropriate.

[69] In step 808, other necessary management services software is installed. This software includes other management facades, such as the data services volume management facade 240 and its accompanying native interface 242 and federated beans such as a configuration manager bean (not shown in Figure 2) and the data service volume bean 230.

[70] Then, in step 810, the Jiro services software is started with the Jiro domain name jiro:Host_a. In step 812, the SNDR and other federated beans are deployed in the Jiro domain. During this step, necessary management facades get automatically instantiated. The process then finishes in step 814.

[71] A process equivalent to that shown in Figure 8 is also carried out on host B 722 in geographic region II, except that the Jiro domain name will be jiro:Host_b for host B.

[72] After the installation and deployment steps are complete in both host A 702 and host B 722, the process of configuring the system can begin. The steps involved in this process are illustrated in Figures 9A-9D. During this process, the

system manager executes two CLI commands, or equivalently, uses the SNDR GUI to generate the two commands. The first command creates and enables an SNDR volume set in both host A 702 and host B 722. The SNDR volume set includes a primary volume and a secondary volume. The second command synchronizes volume B 732 (the secondary volume of the SNDR set) with volume A 720 (the primary volume of the SNDR set). Once the synchronization is complete, the SNDR set will be in REPLICATING mode. From that time, all the input/output information sent to volume A 720 will automatically be replicated to volume B 732, and the content of both the volumes will be the same (that is, "mirrored").

[73] The configuration process begins in step 900 and proceeds to step 902 where, from the command prompt at terminal 700, the system manager issues the following command, or a similar command:

```
sndradm -C p s y n MySNDRSet Host_a Volume_a MetaInfoVolume_a
 Host_b Volume_b MetaInfoVolume_b
```

[74] Alternatively, the command can be generated from information entered into a GUI. In the discussion below, use of the CLI program is assumed. Those skilled in the art would know that the GUI disclosed above could also be used in an equivalent fashion. As set forth in step 904, entry of the command, starts a Java Virtual Machine (JVM) for the SNDR CLI program 904 and passes in necessary information, such as an identification of the host in which the CLI was issued (host A 702), a port number for the Jiro™ service (typically 4160), the Jiro domain name in which the federated beans, including bean 706, and management facades, including management facade 716 are deployed (in this case jiro:Host_a) as well as the SNDR options used in sndradm command.

[75] Next, in step 906, the SNDR CLI program 704 parses the command line options used while invoking the sndradm module. After parsing the options, the CLI program 704 determines that the sndradm module was invoked to create a SNDR

volume set. Since this operation will need to use the SNDR federated bean 704, the CLI program 704 uses a lookup service that is part of the Jiro program to get a proxy handle of the SNDR federated bean 704 that is managing the SNDR data services 718 on host A 702 in the domain jiro:Host_a.

[76] Once the SNDR CLI program 704 locates the appropriate SNDR federated bean 706 and retrieves the proxy handle to the bean 706, in step 908, the CLI program 704 hands over the proxy handle and all other command line arguments passed in for SNDR set creation to a module or subprogram (realized by a Java class) that performs the rest of the specialized steps involved for SNDR set creation.

[77] In step 910, the SNDR CLI set creation module then calls a getSNDRBeanSystemManager() method on the SNDR federated bean proxy handle to retrieve a handle to a SNDRBeanSystemManager object which manages the overall SNDRSystem and provides APIs to create SNDR sets. The SNDR CLI set creation module then informs the SNDRBeanSystemManager of its intention to create a SNDR set using the user-provided parameters for different components of the SNDR set by calling an initializeSet() method of the SNDRBeanSystemManager() object with appropriate arguments.

[78] In the initializeSet() method, in step 912, the SNDRBeanSystemManager informs a data service volume federated bean (not shown in Figure 7) running on host A 702 about volume A 720 in this host that the SNDR federated bean 704 is going to use. The SNDRBeanSystemManager does that by calling a registerSVIVolume() method of the data service volume federated bean with appropriate volume names as arguments in order to register the volumes with the data service volume federated bean.

[79] The process then proceeds, via off-page connectors 914 and 916 to step 918 where, the SNDRBeanSystemManager object calls a getSNDRSystem() method of SNDR bean 704. Inside the getSNDRSystem() method, SNDR bean 704 contacts the SNDR management facade 716 (indicated schematically by arrow 714), retrieves a handle to the SNDRSystem object that the management facade 716 is using as the manager of individual SNDR sets and groups, and returns the handle to the SNDRBeanSystemManager object.

[80] Once the SNDRBeanSystemManager object gets the SNDRSystem handle from the management facade layer 716, in step 920, it asks the SNDRSystem object to create a SNDR set with the user provided data by calling the initializeSet() method of the SNDRSystem object. This latter method call is at the management facade layer 716 and, in the initializeSet() method, the task of set initialization gets delegated to another initialize() method of the SNDRManagerImpl, a module/subprogram implements the management facade SNDRSystem.

[81] In step 922, the SNDRManagerImpl initialize() method first performs several error checks. For example, it checks for the validity of the user provided data and whether the resulting SNDR set will violate any constraints, etc. If all the checks are passed, then a new SNDRSet object is created and added to the list of the SNDR sets managed by this SNDRManagerImpl. At this stage, the newly created SNDR set is in a DISABLED state and the kernel does not yet know of this set.

[82] If the manager indicated in the CLI command that the newly-created SNDR set should be enabled as well during the initialization step itself (as in the command referenced above), then, in step 924, the SNDRManagerImpl object calls an enable() method of this newly-created SNDR set. In this enable() method, a set of native interface objects are created to communicate with the host A kernel. These native interface objects are populated with the appropriate SNDR set information and SNDR data services layer 718 is requested to enable a SNDR set with the information passed through these native objects. The data services kernel layer 718 then enables the set by updating/creating new internal kernel data structures. The enabled set goes into LOGGING mode immediately as host B might, or might not, be ready. (In the situation described now, the other side is actually not ready yet.) In any case, the kernel does not automatically try to enter REPLICATING mode by contacting the other host.

[83] Next, in step 926, after the new SNDR set is created and enabled, a handle to it is returned to the SNDRBeanSystemManager object. The SNDRBeanSystemManager then contacts a configuration manager federated bean running on host A (not shown in Figure 7) in order to indicate that a new SNDR set has

been created. The configuration manager federated stores the information related to this new SNDR set. This configuration information can be retrieved later.

[84] In step 928, the SNDRBeanSystemManger object contacts SNDR federated bean 724 running on host B 722 where the data in volume A 720 will be replicated for this SNDR set. This contact is made, via network 710, as indicated schematically by arrow 708. The process then proceeds, via off-page connectors 930 and 932 to step 934. In step 934, an SNDR set is created and enabled on host B 722. In particular, SNDR federated bean 724 on host B 722 creates and enables an SNDR set on host B 722 by contacting the management facade 730 as indicated by arrow 726 and performing the process similar to that detailed in steps 910 through 926. During this process, the SNDR data service layer 728 is used to create the new SNDR set.

[85] At this point, a new SNDR set has been created and enabled at both hosts 702 and 722 and the CLI command issued in step 902 has been completed. In order to start replication of I/O information destined for primary volume A 720 to the secondary volume B 732, primary volume 720 and secondary volume 732 need to be synchronized. Synchronization is achieved by using another CLI command. In particular, after completion of the first command, in step 936, a system manager issues the following command on terminal 700:

```
sndradm -c s MySNDRSet
```

[86] When the command mentioned above is issued, the CLI program 702 performs several steps similar to steps 904-908. These steps include, starting a Java Virtual Machine for the SNDR CLI program 704 and passing in necessary information (step 938), parsing the command line options and using the Jiro lookup service to get a proxy handle of the SNDR federated bean 704 (step 940) and handing over the proxy handle and all other command line arguments to a synchronization subprogram/module that actually performs the synchronization (step 942.)

[87] In step 944 the synchronization module of the CLI program 704 calls a getSNDRBeanSetManager() method on the proxy handle to the SNDR federated bean 706 to retrieve a handle to the SNDRBeanSetManager object which manages the operations pertaining to individual SNDR sets.

[88] The process then proceeds, via off-page connectors 946 and 948, to step 950 where the synchronization module calls a fullSync() method of the SNDRBeanSetManager object for the appropriate SNDRSet. The fullSync() method of SNDRBeanSetManager object, in turn, calls the getSNDRSystem() method of SNDR bean 706. Inside the getSNDRSystem() method, the SNDR bean 706 contacts the SNDR management facade 716, retrieves a handle to the SNDRSystem object that the management facade 716 is using as the manager of individual SNDR sets and groups, and returns the handle to the SNDRBeanSetManger.

[89] Then, in step 952, the SNDRBeanSetManager object uses the handle to retrieve another handle to the appropriate SNDRSet by calling getSNDRSet() method on the SNDRSystem object. After that, the fullSync() method on the retrieved handle of SNDRSet is called.

[90] In the fullSync() method of the SNDR set, necessary objects in the native interface (not shown in Figure 7) are set up for communication with the SNDR data services layers 718 and 728 in step 954. In step 956, the SNDR data service layer 718 is then requested to synchronize the primary and secondary volumes through these native library objects and method calls. Once the synchronization is complete, the SNDR set goes to the REPLICATING mode and the process ends in step 958. From this time on, all the I/O information destined for the primary volume 720 will automatically be replicated to the secondary volume 732 as well.

[91] A software implementation of the above-described embodiment may comprise a series of computer instructions either fixed on a tangible medium, such as a computer readable media, for example, a diskette, a CD-ROM, a ROM memory, or a fixed disk, or transmittable to a computer system, via a modem or other interface device over a medium. The medium can be either a tangible medium, including but not limited to optical or analog communications lines, or may be implemented with wireless

techniques, including but not limited to microwave, infrared or other transmission techniques. It may also be the Internet. The series of computer instructions embodies all or part of the functionality previously described herein with respect to the invention. Those skilled in the art will appreciate that such computer instructions can be written in a number of programming languages for use with many computer architectures or operating systems. Further, such instructions may be stored using any memory technology, present or future, including, but not limited to, semiconductor, magnetic, optical or other memory devices, or transmitted using any communications technology, present or future, including but not limited to optical, infrared, microwave, or other transmission technologies. It is contemplated that such a computer program product may be distributed as a removable media with accompanying printed or electronic documentation, e.g., shrink wrapped software, pre-loaded with a computer system, e.g., on system ROM or fixed disk, or distributed from a server or electronic bulletin board over a network, e.g., the Internet or World Wide Web.

[92] Although an exemplary embodiment of the invention has been disclosed, it will be apparent to those skilled in the art that various changes and modifications can be made which will achieve some of the advantages of the invention without departing from the spirit and scope of the invention. For example, it will be obvious to those reasonably skilled in the art that, in other implementations, different arrangements can be used for the scope and arrangement of the federated beans. Other aspects, such as the specific process flow, as well as other modifications to the inventive concept are intended to be covered by the appended claims.

[93] What is claimed is: